Journal on Satisfiability, Boolean Modeling and Computation 14 (2023) 1–15

An Isomorphism-Invariant Distance Function on Propositional Formulas in CNF

Evgeny Dantsin edantsin@roosevelt.edu
Alexander Wolpert awolpert@roosevelt.edu
Department of Computer Science, Roosevelt University, IL, USA

Abstract

We define a distance function on propositional formulas in CNF as a measure of non-isomorphism of formulas: the larger the distance between two formulas is, the further they are from being isomorphic. This distance induces a metric on isomorphism classes of formulas. We show how this distance can be used for SAT solving, namely for per-instance algorithm selection where there is a "portfolio" of SAT solvers and there is a "meta-solver" that chooses a solver from the portfolio for a given input formula.

KEYWORDS: *Distance on CNF formulas, isomorphism invariants, per-instance algorithm selection*

Submitted 4 October 2021; revised 9 May 2022; accepted 28 June 2022

1. Introduction

Two propositional formulas in CNF are *isomorphic* if one of them can be obtained from the other by renaming variables and flipping literals. It is natural to view isomorphism as "similarity" of formulas, but this similarity measure is all or nothing: any two formulas are similar or not. How can we modify this measure to make it gradual? In this paper, we define a distance functions on formulas such that the smaller the distance between two formulas is, the closer they are to being isomorphic. This distance function is a pseudometric on formulas and, thereby, it induces a metric on isomorphism classes of formulas.

Application to SAT *Solving.* Before we describe the distance function and its properties, we briefly note how this distance can be used for SAT solving, see more details in this section below and a formal description in Sections 5 and 6. We consider the following variant of *per-instance algorithm selection* [11]. There are a "dataset" $\mathcal{D}$ of formulas and a "portfolio" $\mathcal{P}$ of SAT algorithms such that for every formula $\phi \in \mathcal{D}$, at least one algorithm in $\mathcal{P}$ performs fast on ϕ. There is a "meta-algorithm" $\mathcal{S}$ that takes a formula ψ as input and tries to select an algorithm from $\mathcal{P}$ that performs fast on ψ. How does $\mathcal{S}$ work?

Under certain conditions on $\mathcal{D}$ and $\mathcal{P}$ (we specify these conditions in Section 6), if an algorithm from $\mathcal{P}$ performs fast on a formula $\phi \in \mathcal{D}$ then this algorithm performs fast on any formula close to ϕ. This suggests that $\mathcal{S}$ finds a formula $\phi \in \mathcal{D}$ closest to ψ and selects an algorithm $A \in \mathcal{P}$ that performs fast on ϕ. If the distance between ψ and ϕ is small enough, then A performs fast on ψ as well. Otherwise ($\mathcal{D}$ contains no formula close to ψ), the meta-algorithm $\mathcal{S}$ gives up and returns "can't select".

Distance Function. Precise definitions are given in Sections 2 and 3; here we give the main idea. The distance between two formulas is defined by combining isomorphism of formulas and symmetric difference of sets. Let ϕ_1 and ϕ_2 be propositional formulas in CNF (ϕ_1 and ϕ_2 are

sets of clauses). The distance between them, denoted $\delta(\phi_1, \phi_2)$, is the minimum cardinality of the symmetric differences of formulas ϕ_1' and ϕ_2', where the minimum is taken over all formulas ϕ_1' and ϕ_2' such that they are isomorphic to ϕ_1 and ϕ_2 respectively. This definition has two equivalent forms:

- $\delta(\phi_1, \phi_2)$ is the minimum number of clauses that can be removed from ϕ_1 and ϕ_2 so that the remaining subformulas are isomorphic to each other;
- $\delta(\phi_1, \phi_2)$ is the minimum number of clauses that can be added to ϕ_1 and ϕ_2 so that the resulting superformulas are isomorphic to each other.

By definition, δ is preserved under isomorphism: if ϕ_1, ϕ_1' and ϕ_2, ϕ_2' are pairs of isomorphic formulas, then

$$\delta(\phi_1, \phi_2) = \delta(\phi_1', \phi_2').$$

In Section 3, we show that δ is a pseudometric on formulas. This pseudometric induces a metric on isomorphism classes of formulas.

Note that this approach to defining a distance function is essentially a replica of the approach used by Gromov in the definition of the Gromov–Hausdorff distance on metric spaces [5,10]. The Gromov–Hausdorff distance between metric spaces M_1 and M_2 is the minimum Hausdorff distance between the images of M_1 and M_2 under distance-preserving functions f_1 and f_2 from these spaces to a metric space M, where the minimum is taken over all M, f_1, and f_2.

Computing the Distance. In Section 4 we address the problem of computing the distance function δ. Although it is NP-hard to compute $\delta(\phi_1, \phi_2)$ in general, it is easier to determine whether this distance is "small". Namely, for every integer d, the problem of verifying the equality $\delta(\phi_1, \phi_2) = d$ is Turing reducible to the graph isomorphism problem. The latter can be solved in quasi-polynomial time [4] and, in practice, it is often solved efficiently using Nauty tools [14].

What distance is computed in the per-instance algorithm selection approach described above? The meta-algorithm determines whether the dataset contains a formula lying within a small distance from the input formula ψ. Assuming that a "small distance" means a constant distance, this can be done by a polynomial-time oracle algorithm with an oracle for graph isomorphism.

Per-Instance Algorithm Selection. Section 6 shows how the distance δ can be used for per-instance algorithm, see the brief sketch above. The meta-algorithm $\mathcal{S}$ defined in this section is called the *fixed-distance selector*. Taking a formula ψ and an integer d as input, $\mathcal{S}$ checks whether the dataset $\mathcal{D}$ contains a formula ϕ such that $\delta(\phi, \psi) \leqslant d$. If so, $\mathcal{S}$ selects an algorithm $A \in \mathcal{P}$ that performs fast on ϕ and returns it. Otherwise, $\mathcal{S}$ says "can't select". We formulate conditions on $\mathcal{D}$ and $\mathcal{P}$ that guarantee fast performance of A on the input formula ψ.

The key point of this approach is that every algorithm $A \in \mathcal{P}$ has the following property:

If A performs fast on a formula ϕ, then A performs fast on any formula close to ϕ.

To formalize this property, we define a class of parameters of formulas that we call *parameters of constant variation* (Section 5). By a *parameter* we mean any computable function from the set of formulas to the non-negative integers. A parameter π is a *parameter of constant variation* if the following holds:

- π is preserved under isomorphism;
- for every formula ϕ, a slight variation of ϕ can cause only a slight change of $\pi(\phi)$: the addition of one clause to ϕ can change $\pi(\phi)$ by only a constant value that does not depend on ϕ.

The class of constant-variation parameters includes incidence treewidth, maximum deficiency, and many other parameters π such that the parameterized problem SAT(π) is fixed-parameter tractable [15]. This class can be characterized in terms of the distance δ. Namely, π is a parameter of constant variation if and only if

$$\left|\pi(\phi_1) - \pi(\phi_2)\right| \leqslant c \cdot \delta(\phi_1, \phi_2)$$

for some number c and all formulas ϕ_1 and ϕ_2. This characterization connects the running time of parameterized algorithms from the portfolio $\mathcal{P}$ with the distance δ.

Extension to CSP Instances. In Section 7 we briefly discuss an extension of the distance function δ from CNF formulas to instances of the constraint satisfaction problem.

2. Formulas and Isomorphism

Propositional Formulas in CNFs. A *literal* is a propositional variable or its negation; each of them is the *complement* of the other. The complement of a literal a is denoted by $\neg a$. A *clause* is a nonempty finite set of literals that contains no pair of complements. A *propositional formula in Conjunctive Normal Form*, called a *formula* or *CNF formula* for short, is a finite set of clauses.

Let ϕ be a formula. We write $|\phi|$ to denote the number of clauses of ϕ and we write $\mathrm{var}(\phi)$ to denote the set of variables appearing in ϕ (with or without negation). The *width* of a clause C is the number of literals in C. If ψ is a formula such that $\phi \subseteq \psi$, we call ϕ a *subformula* of ψ.

An *assignment* for a formula ϕ is a function from $\mathrm{var}(\phi)$ to $\{0, 1\}$. Let α be such an assignment and $x \in \mathrm{var}(\phi)$. We say that x is *true* under α if $\alpha(x) = 1$; otherwise x is *false* under α. Symmetrically, we say that $\neg x$ is *true* under α if $\alpha(x) = 0$; otherwise $\neg x$ is *false* under α. A clause is thought of as the disjunction of its literals; we say that α *satisfies* a clause $C \in \phi$ if at least one literal in C is true under α. A formula is thought of as the conjunction of its clauses; if α satisfies all clauses of ϕ then we say that α *satisfies* ϕ and call α a *satisfying assignment* for ϕ. The empty formula, denoted by $\top$, is considered to be satisfied by any assignment. The number of satisfying assignments for ϕ is denoted by $\#\phi$ [9].

We write SAT to denote the satisfiability problem for propositional formulas in CNF: given a formula, does it have a satisfying assignment? The counting version of SAT, denoted by #SAT, is the following function problem: given a formula ϕ, find $\#\phi$.

Isomorphisms. Informally, two formulas are *isomorphic* if one of them can be obtained from the other by renaming variables and flipping literals. A more formal definition is given below.

Let ϕ_1 and ϕ_2 be formulas. Let L_1 and L_2 be the sets of literals defined by

$$L_1 = \mathrm{var}(\phi_1) \cup \{\neg x \mid x \in \mathrm{var}(\phi_1)\}$$

$$L_2 = \mathrm{var}(\phi_2) \cup \{\neg x \mid x \in \mathrm{var}(\phi_2)\}$$

Let f be a bijection from L_1 to L_2. We call f an *isomorphism* from ϕ_1 to ϕ_2 if the following holds:

- f and its inverse preserve the complement relation, which means that for all literals $a, b \in L_1$,

$$a \text{ is the complement of } b \quad \Leftrightarrow \quad f(a) \text{ is the complement of } f(b);$$

- f and its inverse preserve the clauses, which means that for all literals $a_1, \ldots, a_k \in L_1$,

$$\{a_1, \ldots, a_k\} \text{ is a clause of } \phi_1 \quad \Leftrightarrow \quad \{f(a_1), \ldots, f(a_k)\} \text{ is a clause of } \phi_2.$$

Formulas ϕ_1 and ϕ_2 are *isomorphic*, written as $\phi_1 \simeq \phi_2$, if there is an isomorphism from ϕ_1 to ϕ_2. For every formula ϕ, we write $[\phi]$ to denote the *isomorphism class* of ϕ, i.e., the set of all formulas isomorphic to ϕ. Thus, $[\phi]$ can be viewed as ϕ up to isomorphism. Note that an isomorphism between formulas is sometimes defined using renamings only [3], yet in this paper we adhere to the more general version.

3. Distance Function on Formulas

Let X be a set. A function d from $X \times X$ to $\mathbb{R}_{\geqslant 0}$ is a *distance function* if $d(x, y) = d(y, x)$ and $d(x, x) = 0$ for all $x, y \in X$. Consider the following two conditions on d: for all $x, y, z \in X$,

(1) if $d(x, y) = 0$ then $x = y$;
(2) $d(x, y) \leqslant d(x, z) + d(z, y)$ (triangle inequality).

The distance function d is a *metric* on X if d satisfies both (1) and (2). If d satisfies (2), but not necessarily (1), then d is a *pseudometric* on X.

To define a distance function on formulas, we recall the notation for the symmetric difference operation. The *symmetric difference* of sets A and B is denoted by $A \triangle B$:

$$A \triangle B = (A \setminus B) \cup (B \setminus A) = (A \cup B) \setminus (A \cap B).$$

Definition 1. Let δ be the following function that maps pairs of CNF formulas to nonnegative integers:

$$\delta(\phi_1, \phi_2) = \min_{\phi_1', \phi_2'} |\phi_1' \triangle \phi_2'|$$

where the minimum is taken over all formulas ϕ_1' and ϕ_2' such that $\phi_1' \simeq \phi_1$ and $\phi_2' \simeq \phi_2$.

Proposition 1. *The distance function δ is a pseudometric.*

Proof: Consider the distance function $d_\triangle$ on finite sets defined as follows: for all finite sets A and B,

$$d_\triangle(A, B) = |A \triangle B|.$$

It is well known that $d_\triangle$ is a metric. Indeed, we have

$$(A \triangle B) \triangle (B \triangle C) = A \triangle C$$

and, hence, the triangle inequality for $d_{\triangle}$ holds. Now the triangle inequality for δ follows from the triangle inequality for $d_{\triangle}$. □

Suprema and Infima. We say that a formula ϕ is *embedded* into a formula ψ if ϕ is isomorphic to some subformula of ψ. A formula ψ is called a *supremum* of formulas ϕ_1 and ϕ_2 if the following holds:

- both ϕ_1 and ϕ_2 are embedded into ψ;
- for every formula ψ' such that both ϕ_1 and ϕ_2 are embedded into ψ', we have $|\psi| \leqslant |\psi'|$.

Similarly, we say that a formula θ is an *infimum* of formulas ϕ_1 and ϕ_2 if

- θ is embedded into both ϕ_1 and ϕ_2;
- for every formula θ' such that θ is embedded into both ϕ_1 and ϕ_2, we have $|\theta| \geqslant |\theta'|$.

Proposition 2. *Let ψ and θ be respectively a supremum and an infimum of formulas ϕ_1 and ϕ_2. Then*

$$\delta(\phi_1, \phi_2) = |\psi| - |\theta|; \tag{1}$$

$$\delta(\phi_1, \phi_2) = 2|\psi| - |\phi_1| - |\phi_2|; \tag{2}$$

$$\delta(\phi_1, \phi_2) = |\phi_1| + |\phi_2| - 2|\theta|. \tag{3}$$

Proof: It straightforwardly follows from the definition of δ and properties of the symmetric difference. □

Induced Metric on Isomorphism Classes. It follows from the definition of δ that if $\phi_1 \simeq \phi_1'$ and $\phi_2 \simeq \phi_2'$, then

$$\delta(\phi_1, \phi_2) = \delta(\phi_1', \phi_2').$$

Thus, δ induces a distance function on isomorphism classes of formulas: this function, denoted by Δ, is defined by

$$\Delta\big([\phi_1], [\phi_2]\big) = \delta(\phi_1, \phi_2).$$

Proposition 1 implies that Δ is a metric on the set of isomorphism classes.

Example. Let ϕ_1 and ϕ_2 be the following formulas:

$$\phi_1 = \{\{x_1, \neg x_4\}, \{x_2, \neg x_3\}, \{\neg x_1, \neg x_2, x_4\}\};$$
$$\phi_2 = \{\{y_1, \neg y_2, y_3\}, \{\neg y_1, y_2\}, \{y_3, y_4, \neg y_5\}, \{\neg y_4\}\}.$$

Consider the following subformula of ϕ_1:

$$\theta = \{\{x_1, \neg x_4\}, \{\neg x_1, \neg x_2, x_4\}\}.$$

The formula θ is isomorphic to the following subformula of ϕ_2:

$$\{\{y_1, \neg y_2, y_3\}, \{\neg y_1, y_2\}\}$$

(this isomorphism is given by $x_1 \mapsto \neg y_1$, $x_2 \mapsto \neg y_3$, $x_4 \mapsto \neg y_2$). Thus, θ is embedded into both ϕ_1 and ϕ_2. It is easy to check that no formula with more than two clauses is embedded into ϕ_1 and ϕ_2. Hence, θ is an infimum of ϕ_1 and ϕ_2. Using Proposition 2, namely equality (3), we can find the distance between ϕ_1 and ϕ_2:

$$\delta(\phi_1, \phi_2) = |\phi_1| + |\phi_2| - 2|\theta| = 3 + 4 - 2 \cdot 2 = 3.$$

The distance can also be found using a supremum. Consider the following formula that contains ϕ_2 as a subformula:

$$\psi = \{\{y_1, \neg y_2, y_3\}, \{\neg y_1, y_2\}, \{y_3, y_4, \neg y_5\}, \{\neg y_4\}, \{\neg y_3, \neg y_4\}\}.$$

The formula ϕ_1 is embedded into ψ with the isomorphism given by $x_1 \mapsto \neg y_1$, $x_2 \mapsto \neg y_3$, $x_3 \mapsto y_4$, $x_4 \mapsto \neg y_2$. Since ϕ_1 and ϕ_2 cannot be embedded into a formula with less than five clauses, ϕ is a supremum of ϕ_1 and ϕ_2. By equality (2), we have

$$\delta(\phi_1, \phi_2) = 2|\psi| - |\phi_1| - |\phi_2| = 2 \cdot 5 - 3 - 4 = 3.$$

4. Distance Computation

Given formulas ϕ_1 and ϕ_2, how hard is it to compute $\delta(\phi_1, \phi_2)$? We show that the problem of computing the distance δ is at least as hard as the hardest problems in NP. On the positive side, this general problem can be restricted to a more tractable case: determine whether the distance $\delta(\phi_1, \phi_2)$ is "small", where "small" means a constant not depending on the input formulas.

To make these claims more precise and to prove them, we consider the following computational problems:

- The *embedding problem*: given formulas ϕ_1 and ϕ_2, determine whether ϕ_1 is embedded into ϕ_2.
- The *distance verification problem*: given formulas ϕ_1, ϕ_2, and an integer d, determine whether $\delta(\phi_1, \phi_2) = d$.
- The *distance computation problem*: given formulas ϕ_1 and ϕ_2, find the distance $\delta(\phi_1, \phi_2)$.

Proposition 3. *The embedding problem is* NP*-complete.*

Proof: It is obvious that the embedding problem is in NP. To prove the completeness, we reduce the Hamiltonian path problem to the embedding problem. Without loss of generality, we can assume that instances of the Hamiltonian path problem are graphs without isolated vertices.

In our reduction, every graph G without isolated vertices is represented by a formula denoted by ϕ_G and defined as follows. Let G be a graph on vertices $v_1, \ldots, v_n$ and with edges $e_1, \ldots, e_m$. The formula ϕ_G has $n + m$ variables

$$x_1, \ldots, x_n, y_1, \ldots, y_m$$

where each x_i represents the vertex v_i and each y_k represents the edge e_k. Using these variables, we form m clauses of ϕ_G: for each edge e_k with endpoints v_i and v_j, there is the clause

$\{y_k, x_i, x_j\}$. Note that the map $G \mapsto \phi_G$ has the following property: for all graphs G and H without isolated vertices, H is isomorphic to a subgraph of G if and only if ϕ_H is embedded into ϕ_G.

Now we define a reduction of the Hamiltonian path problem to the embedding problem as follows. Let G be a graph with n vertices (all of them are non-isolated) and m edges. Let H be a Hamiltonian path on n vertices: a graph on vertices $u_1, \ldots, u_n$ with edges between u_i and u_{i+1} where $i = 1, \ldots, n-1$. The reduction maps G to the pair (ϕ_H, ϕ_G). It is easy to see that G has a Hamiltonian path if and only if H is isomorphic to a subgraph of G. Therefore, G has a Hamiltonian path if and only if ϕ_H is embedded into ϕ_G. It remains to note that the pair (ϕ_H, ϕ_G) can be computed from G in polynomial time. □

Proposition 4. *The distance verification problem is* NP*-hard.*

Proof: We show that the embedding problem is reducible to the distance verification problem. The reduction is a polynomial-time algorithm that takes an instance (ϕ_1, ϕ_2) of the embedding problem as input and outputs the equality

$$\delta(\phi_1, \phi_2) = |\phi_2| - |\phi_1|. \tag{4}$$

If ϕ_1 is embedded into ϕ_2 then $\delta(\phi_1, \phi_2) = |\phi_2| - |\phi_1|$ by Proposition 2 and, hence, (4) holds.

Conversely, suppose that (4) holds. By Definition 1,

$$\delta(\phi_1, \phi_2) = |\phi_1' \triangle \phi_2'|$$

for some formulas ϕ_1' and ϕ_2' such that $\phi_1' \simeq \phi_1$ and $\phi_2' \simeq \phi_2$. Hence, using (4), we obtain

$$|\phi_1' \triangle \phi_2'| = |\phi_1'| - |\phi_2'|.$$

By properties of the symmetric difference, the latter equality holds only if ϕ_1' is a subformula of ϕ_2'. Therefore, ϕ_1 is embedded into ϕ_2, which completes the proof. □

Corollary 5. *The distance computation problem is at least as hard as the hardest problems in* NP*.*

Proof: There is a trivial polynomial-time Turing reduction of the distance verification problem (shown to be NP-hard) to the distance computation problem. □

Proposition 6. *The distance verification problem can be solved by an oracle algorithm A with the following properties:*

- *the oracle of A answers questions of whether two given formulas are isomorphic;*
- *there exists a polynomial p such that on every instance (ϕ_1, ϕ_2, d), the running time of A is at most $p(s) \cdot M^d$, where s is the size of the instance and $M = \max(|\phi_1|, |\phi_2|)$.*

Proof: The distance verification problem can be solved using the equivalent definitions of δ given by Proposition 2. For instance, we can use equality (3) that expresses δ in terms of infima:

$$\delta(\phi_1, \phi_2) = |\phi_1| + |\phi_2| - 2|\theta|$$

where θ is an infimum of ϕ_1 and ϕ_2. In fact, this definition shows that distance verification is equivalent to "infimum verification":

$$\delta(\phi_1, \phi_2) = d \quad \Leftrightarrow \quad |\theta| = \frac{|\phi_1| + |\phi_2| - d}{2}$$

That is, instead of verifying whether $\delta(\phi_1, \phi_2) = d$, we can verify whether ϕ_1 and ϕ_2 have an infimum θ that contains k clauses where

$$k = \frac{|\phi_1| + |\phi_2| - d}{2}.$$

The algorithm A takes (ϕ_1, ϕ_2, d) as input and uses brute force to check whether such an infimum exists. Namely, A takes the following two steps.

First, A enumerates pairs (θ_1, θ_2), where θ_1 is a k-subset of ϕ_1 and θ_2 is a k-subset of ϕ_2. For each such pair, A asks the oracle whether θ_1 and θ_2 are isomorphic. If the oracle's answer is positive, an infimum of cardinality at least k exists. The number of queries to the oracle is estimated by

$$\begin{aligned}\binom{|\phi_1|}{k} \cdot \binom{|\phi_2|}{k} &= \binom{|\phi_1|}{|\phi_1| - k} \cdot \binom{|\phi_2|}{|\phi_2| - k} \\ &\leqslant M^{|\phi_1|-k} \cdot M^{|\phi_2|-k} = M^{|\phi_1|+|\phi_2|-2k} = M^d.\end{aligned}$$

Second, in a similar way, A checks whether there exist a $(k+1)$-subset of ϕ_1 and a $(k+1)$-subset of ϕ_2 such that these subsets are isomorphic. If no such subsets exist, there is an infimum of cardinality k. The total number of queries to the oracle in the two steps does not exceed $2M^d$, and each query is processed in polynomial time. Therefore, the overall running time is at most $p(s) \cdot M^d$ for some polynomial p. □

The algorithm in Proposition 6 uses an oracle for deciding the *formula isomorphism problem* (given two formulas, are they isomorphic?). It is well known that this problem is polynomially equivalent to the graph isomorphism problem [3]. It is easy to modify the algorithm A so that its oracle for the formula isomorphism problem can be replaced with an oracle for the graph isomorphism problem. Let A_G be the modified algorithm; the running time of A_G remains the same as up to a polynomial factor.

The complexity class GI consists of problems that have polynomial-time Turing reduction to the graph isomorphism problem, see for example [12]. It is known that the graph isomorphism problem can be solved in quasi-polynomial time [4] and, in practice, it is often solved efficiently using Nauty tools [14].

Proposition 7. *For every integer d, the restriction of the distant verification problem to instances (ϕ_1, ϕ_2, d) is in* GI.

Proof: For each restriction, the algorithm A_G described above is a polynomial-time Turing reduction to the graph isomorphism problem. □

5. Parameters of Constant Variation

By a *parameter* of formulas we mean any computable function from the set of formulas to the non-negative integers. Natural examples of parameters are the number of clauses, the number of variables, and the number of satisfying assignments. In this section we define a class of *parameters of constant variation*. This type of parameters is needed for the next section where we describe how the distance δ can be used for SAT solving. We give only two examples of constant-variation parameters, namely incidence treewidth and maximum deficiency, but many parameters for which the corresponding parameterized version of SAT is fixed-parameter tractable are parameters of constant variation.

Definition 2 (parameter of constant variation)**.** We say that a parameter π is a *parameter of constant variation* if it has the following two properties:

- for all formulas ϕ_1 and ϕ_2, if $\phi_1 \simeq \phi_2$ then $\pi(\phi_1) = \pi(\phi_2)$;
- there exists a number c such that for every formula ϕ and every formula ϕ' obtained from ϕ by adding one clause, we have

$$|\pi(\phi') - \pi(\phi)| \leqslant c. \tag{5}$$

The number c is called a *bound on the change*.

Thus, π is a parameter of constant variation if it is preserved under isomorphism and the addition of one clause to ϕ can change $\pi(\phi)$ by only a constant value that does not depend on ϕ. The number of clauses in a formula is a trivial example of parameters of constant variation. As for the number of variables, it is not a parameter of constant variation. Indeed, consider formulas ϕ and ϕ' where ϕ' is obtained from ϕ by adding one clause with arbitrarily many new variables. Then there is no constant upper bound on the difference between $|\operatorname{var}(\phi')|$ and $|\operatorname{var}(\phi)|$.

For more illustration of the notion of constant-variation parameters, we give two examples: incidence treewidth and maximum deficiency. Both parameters are well known in connection with fixed-parameter tractability of SAT.

Incidence Treewidth. For every formula ϕ, the *incidence graph* of ϕ is a bipartite graph G defined as follows. The set of vertices of G is partitioned into two subsets: one of them is the set of variables of ϕ and the other is the set of clauses of ϕ. There are only edges between variables and clauses: a variable x is connected by an edge with a clause C if and only if $x \in \operatorname{var}(C)$. The treewidth of the incidence graph of ϕ is called the *incidence treewidth* of ϕ and is denoted by $\operatorname{tw}(\phi)$.

There are various algorithms that solve SAT and #SAT when an input formula is given along with a tree decomposition of its incidence graph. For example, the algorithm from [16] solves #SAT in time $2^w \cdot \operatorname{poly}(l)$ where w is the width of the input tree decomposition and l is the size of the input. Also there are algorithms that build a tree decomposition of a given graph. One of such algorithm is given in [1]: it approximates a tree decomposition with the minimum width. The combination of these two algorithms solves #SAT in time $2^{3.7 \cdot \operatorname{tw}(\phi)} \cdot \operatorname{poly}(l)$.

Proposition 8. *Incidence treewidth is a parameter of constant variation.*

Proof: If two formulas are isomorphic then their incidence graphs are isomorphic too and, therefore, incidence treewidth is preserved under isomorphism. We prove that incidence treewidth has the second property from Definition 2 with $c = 1$: for every formula ϕ and every formula ϕ' obtained from ϕ by adding one clause, we have $\mathrm{tw}(\phi') - \mathrm{tw}(\phi) \leqslant 1$.

Consider a tree decomposition of the incidence graph of ϕ: a labeled tree T in which every vertex is labeled by a bag of vertices. We show how to extend it to a tree decomposition of the incidence graph of ϕ'. Let C be the "new" clause added to ϕ and let $x_1, \ldots, x_k$ be the "new" variables, i.e., the variables appearing in C and not occurring in ϕ. These "new" elements must be added to T. The variables $x_1, \ldots, x_k$ are added in a simple way: (1) select any leaf of T, (2) connect it with k new vertices, and (3) assign one-element bags $\{x_1\}, \ldots, \{x_k\}$ to the new vertices. Finally, we extend all bags (including the bags of new vertices) by adding the clause C.

It is easy to see that the result of our extension, the labeled tree T', is a tree decomposition of the incidence graph of ϕ'. Moreover, by our construction, we have $w' - w = 1$ where w and w' are the width of T and the width of T' respectively. Taking the tree decomposition T such that $w = \mathrm{tw}(\phi)$ and using the obvious inequality $\mathrm{tw}(\phi') \leqslant w'$, we obtain

$$\mathrm{tw}(\phi') - \mathrm{tw}(\phi) \leqslant w' - w = 1,$$

which completes the proof. □

Maximum Deficiency. The parameter of formulas called *maximum deficiency* was introduced in [8]; it relates satisfiability of a formula with matchings in its incidence graph.

Let ϕ be a formula and let M be a matching in its incidence graph. For each edge in M that connects a variable x and a clause C, we can assign x a truth value that makes C true. Therefore, if all clauses of ϕ are matched by M then ϕ is satisfiable. The *maximum deficiency* of a formula ϕ, denoted by $\mathrm{md}(\phi)$, is the number of clauses unmatched by a maximum matching in the incidence graph of ϕ (this number remains the same whatever maximum matching is taken). Since a maximum matching can be found in polynomial time, $\mathrm{md}(\phi)$ is computable in polynomial time. Equivalently, due to Hall's marriage theorem, maximum deficiency can be defined as follows:

$$\mathrm{md}(\phi) = \max_{\phi' \subseteq \phi} \bigl(|\phi'| - |\mathrm{var}(\phi')|\bigr)$$

where the maximum is taken over all subformulas of ϕ. Note that $\mathrm{md}(\phi) \geqslant 0$ for all ϕ because $\top$ is a subformula of every formula. The satisfiability problem parameterized by maximum deficiency is fixed-parameter tractable: the algorithms in [7,17] solve SAT in time $2^{\mathrm{md}(\phi)} \cdot \mathrm{poly}(l)$ where l is the size of ϕ.

Proposition 9. *Maximum deficiency is a parameter of constant variation.*

Proof: First, maximum deficiency is preserved under isomorphism. Second, it is obvious that the addition of one clause to a formula ϕ can increase $\mathrm{md}(\phi)$ by at most one and, therefore, the second property from Definition 2 holds with $c = 1$. □

Characterization in Terms of Distance. The proposition below gives an equivalent definition of parameters of constant variation in terms of the distance δ.

Proposition 10. *The following two statements are equivalent:*

(1) *A parameter π is a parameter of constant variation and c is a bound on the change.*
(2) *Let π be a parameter and let c be a number such that for all formulas ϕ_1 and ϕ_2,*

$$\big|\pi(\phi_1) - \pi(\phi_2)\big| \leqslant c \cdot \delta(\phi_1, \phi_2). \tag{6}$$

Proof: First, suppose that (6) holds. Then $\delta(\phi_1, \phi_2) = 0$ implies $\pi(\phi_1) = \pi(\phi_2)$, which means that π is preserved under isomorphism. If ϕ_2 is obtained from ϕ_1 by adding one clause, then $\delta(\phi_1, \phi_2) = 1$ and, therefore, we have the second property from Definition 2:

$$\big|\pi(\phi_2) - \pi(\phi_1)\big| \leqslant c.$$

Second, suppose that π is a parameter of constant variation and c is a bound on the change. Let ϕ_1 and ϕ_2 be arbitrary formulas. By the definition of δ and Proposition 2, there exist four formulas ϕ_1', ϕ_2', ψ, θ such that

- $\phi_1 \simeq \phi_1'$ and $\phi_2 \simeq \phi_2'$;
- θ is subformula of both ϕ_1' and ϕ_2';
- both ϕ_1' and ϕ_2' are subformulas of ψ;
- $\delta(\phi_1, \phi_2) = |\psi| - |\theta|$.

Thus, ψ is obtained from θ by the addition of $\delta(\phi_1, \phi_2)$ clauses. Since the addition of one clause can change $\pi(\theta)$ by at most c, we have inequality (6). □

Note that inequality (6) is the same as in the definition of *Lipschitz functions* (also called *Lipschitz continuous* functions or *Lipschitz maps*) [5]. Thus, loosely speaking, π is a parameter of constant variation if and only if π is a Lipschitz function.

6. Fixed-Distance Selector

There are many techniques for designing a "meta-algorithm" that solves SAT using a given "portfolio" of SAT algorithms [11]. We show how the distance δ can be used in a variant of the *per-instance algorithm selection* technique. Namely, we describe an incomplete meta-algorithm, called a *fixed-distance selector*, that takes a formula ψ as input and tries to select an algorithm from the portfolio that performs fast on ψ. Then we formulate conditions on portfolios and prove that if the conditions are met then the selected algorithm indeed performs efficiently on ψ.

Datasets and Portfolios. We consider algorithms that take formulas as inputs, for example, algorithms that solve SAT, or #SAT, or MAXSAT. Let $\mathcal{D}$ be a finite set of formulas called a *dataset*. With every formula $\phi \in \mathcal{D}$, we associate a non-empty finite set E_ϕ of algorithms that perform fast on ϕ (we do not define here what "fast" means, this will be done in the conditions formulated below). Note that sets E_ϕ associated with different formulas may or may not overlap. Given a dataset $\mathcal{D}$, the set

$$\mathcal{P} = \bigcup_{\phi \in \mathcal{D}} E_\phi$$

of algorithms is called the *portfolio* corresponding to $\mathcal{D}$. Thus, for every formula $\phi \in \mathcal{D}$, there is at least one algorithm $A \in \mathcal{P}$ that performs fast on ϕ. Likewise, every algorithm in the portfolio performs fast on at least one formula from the dataset.

Fixed-Distance Selection. We call the following simple meta-algorithm the *fixed-distance selector* and denote it by $\mathcal{S}$. The algorithm uses a dataset $\mathcal{D}$ and the corresponding portfolio $\mathcal{P}$. Let ψ be a formula and let d be a non-negative integer. Taking ψ and d as input, the selector $\mathcal{S}$ either outputs an algorithm $A \in \mathcal{P}$ or returns "can't select":

(1) Find a formula $\phi \in \mathcal{D}$ such that $\delta(\phi, \psi) \leqslant d$ and $\delta(\phi, \psi)$ is the minimum over all formulas $\phi \in \mathcal{D}$. If there is no such formula ϕ, return "can't select".
(2) Select any algorithm A from E_ϕ and return it.

We do not specify how to find the formulas of $\mathcal{D}$ closest to ψ and how to choose one of them. The straightforward way of finding ϕ is exhaustive search: for $r = 0, 1, \ldots, d$, enumerate all formulas $\phi \in \mathcal{D}$ and verify whether $\delta(\phi, \psi) = r$ (the distance verification problem). It follows from Proposition 6 that each verification can be done using a polynomial-time oracle algorithm with an oracle for graph isomorphism. Thus, this exhaustive search requires at most $(d+1)N$ runs of the oracle algorithm where N is the cardinality of the dataset.

What is the role of the input number d, should it be small or large? Proposition 11 gives an upper bound on the running time of the selected algorithm A on ψ. This bound depends on d: the smaller d is, the faster A is. On the other hand, assuming that N is fixed, if d is too small, this increases the chance that $\mathcal{S}$ returns "can't select". Thus, a good tradeoff for choosing d is needed.

Algorithms Matching with Parameters. An efficient performance of an algorithm A on a given class of formulas is typically due to the fact that A makes use of some "hidden structure" of formulas of this class. Different algorithms exploit different hidden structures and, thereby, they are successful on different classes of formulas. Hidden structures can be utilized explicitly or implicitly. For example, the fixed-parameter tractable algorithms mentioned in Section 5 make explicit use of certain properties of the incidence graph: the existence of a tree decomposition with a small treewidth and a small maximum deficiency. The CDCL SAT solvers in [2,13] can be viewed as examples of implicit use of community structures.

A hidden structure is often characterized by a parameter π such that the value $\pi(\psi)$ determines how fast an algorithm A runs on ψ: the smaller $\pi(\psi)$ is, the faster A on ψ is. For the most of such parameters, the running time of A on ψ is exponential in $\pi(\psi)$ and polynomial in the size of ψ [6,15]. We say that A *matches* with π if there exist a number b and a polynomial q such that for every formula ψ, the running time of A on ψ is at most $b^{\pi(\psi)} \cdot q(l_\psi)$ where l_ψ is the size of ψ.

Note two differences between an algorithm A matching with a parameter π and a fixed-parameter algorithm. First, the input to A is a formula, while the input to a fixed-parameter algorithm is a formula along with a value of the parameter. Second, the upper bound for A has the exponential factor $b^{\pi(\psi)}$ instead of an arbitrary function $f(\pi(\psi))$ as in the case of a fixed-parameter algorithm.

Conditions on Datasets and Portfolios. Let $\mathcal{D}$ be a dataset and $\mathcal{P}$ be the corresponding portfolio. Let a, b, and c be numbers and let q be a polynomial. We say that the tuple (a, b, c, q) is a *bound* for $\mathcal{D}$ and $\mathcal{P}$ if for every formula $\phi \in \mathcal{D}$ and every algorithm $A \in E_\phi$, there exists a parameter π such that the following three conditions are met:

- *Condition 1.* $\pi(\phi) \leqslant a$.

- *Condition 2.* The algorithm A matches with π and the corresponding base and polynomial are bounded from above by b and q respectively: for every formula ψ, the running time of A on ψ does not exceed $b^{\pi(\psi)} \cdot q(l_\psi)$ where l_ψ is the size of ψ.
- *Condition 3.* The parameter π is a parameter of constant variation and the corresponding bound on the change is less than or equal to c.

Proposition 11. *Let $\mathcal{D}$ be a dataset and $\mathcal{P}$ be the corresponding portfolio used by the fixed-distance selector $\mathcal{S}$ described above. Suppose that numbers a, b, c, and a polynomial q are bounds for $\mathcal{D}$ and $\mathcal{P}$. Given a formula ψ and an integer d as input, let $A \in \mathcal{P}$ be the algorithm returned by $\mathcal{S}$ on this input. Then the running time of A on ψ is at most $b^{a+cd} \cdot q(l_\psi)$ where l_ψ is the size of ψ.*

Proof: Let $\phi \in \mathcal{D}$ be the formula chosen by $\mathcal{S}$ as a formula closest to the input formula ψ. By Conditions 2 and 3, there is a parameter π such that

- π is a parameter of constant variation;
- A runs on ψ in time at most $b^{\pi(\psi)} \cdot q(l_\psi)$.

By Proposition 10 we have $\pi(\psi) \leqslant \pi(\phi) + c \cdot \delta(\phi, \psi)$. By Condition 1 we have $\pi(\phi) \leqslant a$. Therefore, $\pi(\psi) \leqslant a + cd$. □

Thus, A performs efficiently on ψ: assuming that d in the bound is fixed, there exists a polynomial $p(l) = b^{a+cd} \cdot q(l)$ such that the running time of A on ψ does not exceed $p(l_\psi)$.

7. Concluding Remarks: Possible Extensions

Isomorphism of Formulas. There are several equivalent definitions of the distance δ (Proposition 2) and all of them use isomorphism of formulas as the central notion. On the other hand, these definitions do not require exactly our notion of isomorphism: two formulas are isomorphic if one of them can be obtained from the other by renaming variables and flipping literals. For example, we could define $\phi_1 \simeq \phi_2$ based on only renaming, or only flipping, or another equivalence relation on formulas. In fact, we could define the distance *relative to* any equivalence relation on formulas: if I is such an equivalence relation, then δ^I is the corresponding distance function on formulas. This distance δ^I could be of interest for SAT solving if, first, the satisfiability status is preserved under I and, second, the problem of I-equivalence testing is "easier" than the problem of satisfiability testing (like the graph isomorphism problem is supposedly easier than the satisfiability problem).

Extension to CSP Instances. The main definitions and results of this paper can be extended from formulas in CNF to instances of the constraint satisfaction problem (CSP instances). The adjustments are obvious: clauses are replaced with constraints and isomorphism of formulas is replaced with isomorphism of CSP instances.

There are several variants of isomorphism of CSP instances; we describe the most common one. Let V be a finite set of variables that take values from a finite domain D. A *constraint* of arity k is a pair $(\vec{x}, R)$, where $\vec{x}$ is a k-ary tuple of variables from V and R is a k-ary relation on D. A *CSP instance* is a triple $(V, D, \mathcal{C})$ where $\mathcal{C}$ is finite set of constraints over V and D. Two CSP instances $(V_1, D_1, \mathcal{C}_1)$ and $(V_2, D_2, \mathcal{C}_2)$ are *isomorphic* if there are bijections $f : V_1 \to V_2$ and $g : D_1 \to D_2$ such that the set $\mathcal{C}_2$ of constraints is obtained from $\mathcal{C}_1$ by replacing each variable $x \in V_1$ and each value $a \in D_1$ with their images $f(x)$ and $g(a)$ respectively.

Using this variant of isomorphism, the definitions and propositions of Section 3 are easily carried over from formulas in CNF to CSP instances. Thus, the distance between two CSP instances Φ_1 and Φ_2 is the minimum number of constraints that can be removed from Φ_1 and Φ_2 so that the remaining sub-instances are isomorphic to each other. Or, equivalently, this distance is the minimum number of constraints that can be added to Φ_1 and Φ_2 so that the resulting "super-instances" are isomorphic to each other. The definitions and propositions regarding parameters of constant variation, their characterization in terms of the distance, and fixed-distance selection are carried over to CSP instances as well.

Acknowledgement

We thank the anonymous reviewers for their targeted comments that helped us significantly improve the manuscript.

References

[1] E. Amir, Approximation algorithms for treewidth, *Algorithmica* **56**(4) (2010), 448–479. doi:10.1007/s00453-008-9180-4.

[2] C. Ansótegui, M.L. Bonet, J. Giráldez-Cru and J. Levy, Structure features for SAT instances classification, *Journal of Applied Logic* **23** (2017), 27–39. doi:10.1016/j.jal.2016.11.004.

[3] G. Ausiello, F. Cristiano, P. Fantozzi and L. Laura, Syntactic isomorphism of CNF Boolean formulas is graph isomorphism complete, in: *Proceedings of the 21st Italian Conference on Theoretical Computer Science, 2020*, CEUR Workshop Proceedings, Vol. 2756, CEUR-WS.org, 2020, pp. 190–201.

[4] L. Babai, Graph isomorphism in quasipolynomial time [extended abstract], in: *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016*, ACM, 2016, pp. 684–697.

[5] D. Burago, Y. Burago and S. Ivanov, *A Course in Metric Geometry*, Graduate Studies in Mathematics, Vol. 33, American Mathematical Society, 2001.

[6] M. Cygan, F.V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk and S. Saurabh, *Parameterized Algorithms*, Springer, 2015.

[7] H. Fleischner, O. Kullmann and S. Szeider, Polynomial-time recognition of minimal unsatisfiable formulas with fixed clause-variable difference, *Theoretical Computer Science* **289**(1) (2002), 503–516. doi:10.1016/S0304-3975(01)00337-1.

[8] J.V. Franco and A.V. Gelder, A perspective on certain polynomial-time solvable classes of satisfiability, *Discrete Applied Mathematics* **125**(2–3) (2003), 177–214. doi:10.1016/S0166-218X(01)00358-4.

[9] C.P. Gomes, A. Sabharwal and B. Selman, Model counting, in: *Handbook of Satisfiability*, 2nd edn, Frontiers in Artificial Intelligence and Applications, Vol. 336, IOS Press, 2021, pp. 993–1014, Chapter 25.

[10] M. Gromov, *Metric Structures for Riemannian and Non-Riemannian Spaces*, Progress in Mathematics, Vol. 152, Birkhäuser, 1999, Based on the 1981 French original.

[11] H.H. Hoos, F. Hutter and K. Leyton-Brown, Automated configuration and selection of SAT solvers, in: *Handbook of Satisfiability*, 2nd edn, Frontiers in Artificial Intelligence and Applications, Vol. 336, IOS Press, 2021, pp. 481–507, Chapter 12.

[12] J. Köbler, U. Schöning and J. Torán, *The Graph Isomorphism Problem: Its Structural Complexity*, Progress in Theoretical Computer Science, Birkhäuser/Springer, 1993.

[13] C. Li, J. Chung, S. Mukherjee, M. Vinyals, N. Fleming, A. Kolokolova, A. Mu and V. Ganesh, On the hierarchical community structure of practical Boolean formulas, in: *Proceedings of the 24th International Conference on Theory and Applications of Satisfiability Testing, SAT 2021*, Lecture Notes in Computer Science, Vol. 12831, Springer, 2021, pp. 359–376. doi:10.1007/978-3-030-80223-3_25.

[14] B.D. McKay and A. Piperno, Practical graph isomorphism, II, *Journal of Symbolic Computation* **60** (2014), 94–112. doi:10.1016/j.jsc.2013.09.003.

[15] M. Samer and S. Szeider, Fixed-parameter tractability, in: *Handbook of Satisfiability*, 2nd edn, Frontiers in Artificial Intelligence and Applications, Vol. 336, IOS Press, 2021, pp. 693–736, Chapter 17.

[16] F. Slivovsky and S. Szeider, A faster algorithm for propositional model counting parameterized by incidence treewidth, in: *Proceedings of the 23rd International Conference on Theory and Applications of Satisfiability Testing, SAT 2020*, Lecture Notes in Computer Science, Vol. 12178, Springer, 2020, pp. 267–276. doi:10.1007/978-3-030-51825-7_19.

[17] S. Szeider, Minimal unsatisfiable formulas with bounded clause-variable difference are fixed-parameter tractable, *Journal of Computer and System Sciences* **69**(4) (2004), 656–674. doi:10.1016/j.jcss.2004.04.009.